ランダムな内容のテキストファイルを大量に書き出す

2015年12月26日
.NET Framework 4 がサポートされなくなるので、4.5.2や4.6へのアップグレードを推奨します。
公開しているexeは.NET Framework 4.5.2や4.6でも実行可能です。

2015/08/15
Taskクラス使うのを全面的に止めてParallel.Invoke()を使った処理に変更。

動作設定ファイル config.xml に従って、指定のフォルダ群に指定行数のテキストファイルを指定個数作成します。 テキストファイル書き埋めのソースコードの改良版です。

元のプログラムは書き出す文字として1バイト文字のみ対応していましたが、このプログラムではマルチバイト文字の混在も可能です。
また、スレッドを使ってファイル作成処理を多重化しており元プログラムより速く動作する可能性があります。

ダウンロード

randomwriter.zip - .NET Framework 4.0版のEXE 2015/08/15 Taskオブジェクト生成を止めた□config.xmlは本文のサンプルをダウンロードしてください。

使い方

カレントフォルダにある config.xml に必要な指定を行います。

dirThreadCount,threadCountの値についてはプログラムの引数で指定できます。1番目が dirThreadCount □2番目が threadCount です。
※config.xmlで指定した dirThreadCount,threadCount の値を引数で上書きすることが出来ます。

動作設定ファイル

この例では、

- X:\DummyText\A01\ □ X:\DummyText\Z01\ の26個フォルダを作成
- X:\DummyText\A01\ □ X:\DummyText\Z01\ のタイムスタンプを 2015/01/01 00:00:00 ～ 2015/08/31 23:59:59 の間でランダムに設定
- 各フォルダの中にsctext000000.txt □ sctext000999.txt の1000個テキストファイルが作成

されます。

作成されるテキストファイルは

- UTF-8エンコーディング

- 1行の文字数は1024文字
- 改行文字コードは CR+LF
- 使われる文字は“ ぁあぃいぅうぇえぉおかがきぎくぐけげこごさざしじすずせぜそぞただちぢっつづてでとどなにぬねのはばぱひびぴふぶぷへべぺほぼぽまみむめもゃやゅゆょよらりるれろゎわゐゑをんゔゕゖ ”の中からランダムに選択

の内容となります。

テキストファイル作成処理では

- スレッドを使用し4フォルダ同時処理、各フォルダ25ファイル同時作成
- テキストファイルのタイムスタンプを 2015/01/01 00:00:00 ～ 2015/08/31 23:59:59 の間でランダムに設定

されます。

スレッド数＝ threadCount × dirThreadCount

になります。例だと 25 × 4 = 100スレッドとなります。

config.xml

```
<?xml version="1.0"?>
<configBase>

  <!-- 生成ファイルプリフィックス -->
  <filePrifix>sctext</filePrifix>

  <!-- 生成ファイルサフィックス -->
  <fileSuffix>.txt</fileSuffix>

  <!-- 書き出すパス先頭 -->
  <writePath>X:\DummyText</writePath>

  <!-- 書き出すフォルダ一覧 -->
  <basePaths>
    <string>A01</string>
    <string>B01</string>
    <string>C01</string>
    <string>D01</string>
    <string>E01</string>
    <string>F01</string>
    <string>G01</string>
    <string>H01</string>
    <string>I01</string>
    <string>J01</string>
    <string>K01</string>
    <string>L01</string>
    <string>M01</string>
    <string>N01</string>
    <string>O01</string>
    <string>P01</string>
    <string>Q01</string>
```

```
    <string>R01</string>
    <string>S01</string>
    <string>T01</string>
    <string>U01</string>
    <string>V01</string>
    <string>W01</string>
    <string>X01</string>
    <string>Y01</string>
    <string>Z01</string>
  </basePaths>

  <!-- ファイルのエンコーディング -->
  <encodingStr>utf-8</encodingStr>
  <!--
  <encoding>shift_jis</encoding>
  -->

  <!-- フォルダに書き出すファイル個数 999,999個まで -->
  <fileCount>1000</fileCount>

  <!-- 1行の文字数 99,999,999文字まで -->
  <lineWidth>1024</lineWidth>

  <!-- ファイルの行数 99,999,999行まで -->
  <lineCount>1000</lineCount>

  <!-- ファイルの改行文字 -->
  <terminate>&#13;&#10;</terminate>

  <!-- ファイル書き出しに使う文字 -->
  <chars>ぁあぃいぅうぇえぉおかがきぎくぐけげこござさしじすずせぜそぞただちぢっつづてでとどなにぬねのはばぱひびぴふぶぷへべぺほぼぽまみむめもゃやゅゆょよらりるれろゎわゐゑをんゔゕゖ</chars>
  <!--
<chars>0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ/=
.:\~+</chars>
  -->

  <!-- タイムスタンプランダム化を適用する -->
  <useRandomTimestamp>true</useRandomTimestamp>

  <!-- タイムスタンプランダム化適用時の範囲開始日時 -->
  <sDateTime>2015-01-01T00:00:00.000000+09:00</sDateTime>

  <!-- タイムスタンプランダム化適用時の範囲終了日時 -->
  <eDateTime>2015-08-31T23:59:59.999999+09:00</eDateTime>

  <!-- ファイル作成並列処理数 100まで -->
  <threadCount>25</threadCount>

  <!-- ディレクトリ並列処理数 -->
```

```
  <dirThreadCount>4</dirThreadCount>

</configBase>
```

ソースコード

Visual Studio 2010 .NET Framework 4 で動作確認しました。

2015/08/15 の修正点

- コード内でのTaskオブジェクト生成を完全に廃止。
- とってつけたようなコメントは要らぬ、といわれたのでコメントはずした。
- mkFile()メソッド内のRandomオブジェクトがランダムにならなくなってきたのでシード値を与える修正。

make.bat

```
C:\Windows\Microsoft.NET\Framework\v4.0.30319\Csc.exe /noconfig
/nowarn:1701,1702 /nostdlib+ /warn:4 /reference:"C:\Program Files
(x86)\Reference
Assemblies\Microsoft\Framework\.NETFramework\v4.0\Profile\Client\Micros
oft.CSharp.dll" /reference:"C:\Program Files (x86)\Reference
Assemblies\Microsoft\Framework\.NETFramework\v4.0\Profile\Client\mscorl
ib.dll" /reference:"C:\Program Files (x86)\Reference
Assemblies\Microsoft\Framework\.NETFramework\v4.0\Profile\Client\System
.Core.dll" /reference:"C:\Program Files (x86)\Reference
Assemblies\Microsoft\Framework\.NETFramework\v4.0\Profile\Client\System
.Data.DataSetExtensions.dll" /reference:"C:\Program Files
(x86)\Reference
Assemblies\Microsoft\Framework\.NETFramework\v4.0\Profile\Client\System
.Data.dll" /reference:"C:\Program Files (x86)\Reference
Assemblies\Microsoft\Framework\.NETFramework\v4.0\Profile\Client\System
.dll" /reference:"C:\Program Files (x86)\Reference
Assemblies\Microsoft\Framework\.NETFramework\v4.0\Profile\Client\System
.Xml.dll" /filealign:512 /optimize+ /out:randomWriter.exe /target:exe
Program.cs
```

Program.cs

```
using System;
using System.Collections.Generic;
using System.Collections;
using System.Text;
using System.IO;
using System.Xml.Serialization;
using System.Threading;
using System.Threading.Tasks;
```

```
namespace randomWriter
{
    public class configBase
    {
        public String writePath;
        public String filePrifix;
        public String fileSuffix;
        public String chars;
        public String terminate;
        public String encodingStr;
        public String[] basePaths;
        public int fileCount;
        public int lineWidth;
        public int lineCount;
        public int threadCount;
        public int dirThreadCount=1;
        public DateTime sDateTime = DateTime.Now;
        public DateTime eDateTime = DateTime.Now;
        public Boolean useRandomTimestamp = false;
    }

    public class config
    {
        private configBase cfgb;

        private int terminateLen = 2;
        private int maxCharByte = 0;
        private Random r = new Random();

        private Dictionary<String, int> char2ByteMap = new
Dictionary<String, int>();
        private Dictionary<int, String> index2StringMap = new
Dictionary<int, String>();

        public config()
        {
            readConfig("config.xml");
            checkConfig();
        }
        public config(String filename)
        {
            readConfig(filename);
            checkConfig();
        }
        public config(String filename, int dtc, int tc)
        {
            readConfig(filename);
            cfgb.threadCount = tc;
            cfgb.dirThreadCount = dtc;
            checkConfig();
        }
```

```
        private void readConfig(String filename)
        {
            using (FileStream fs = new FileStream(filename,
FileMode.Open, FileAccess.Read))
            {
                XmlSerializer xs = new
XmlSerializer(typeof(configBase));

                cfgb = xs.Deserialize(fs) as configBase;
                fs.Close();
            }
        }
        private void checkConfig()
        {
            /// fileCount,lineCount,lineWidth の設定値補正

            if ((cfgb.fileCount > 999999)   || (cfgb.fileCount < 1))
cfgb.fileCount = 1;
            if ((cfgb.lineCount > 99999999) || (cfgb.lineCount < 1))
cfgb.lineCount = 1;
            if ((cfgb.lineWidth > 99999999) || (cfgb.lineWidth < 1))
cfgb.lineWidth = 1;


            /// terminate で指定された改行文字列の長さ取得

            terminateLen =
Encoding.GetEncoding(cfgb.encodingStr).GetByteCount(cfgb.terminate);


            /// chars で指定された文字列を文字単位に分解して辞書に格納、文字で使
用している最大バイト数確定

            for (int i = 0; i < cfgb.chars.Length; i++)
            {
                index2StringMap[i] = cfgb.chars.Substring(i, 1);
                char2ByteMap[cfgb.chars.Substring(i, 1)] =
Encoding.GetEncoding(cfgb.encodingStr).GetByteCount(cfgb.chars.Substrin
g(i, 1));
                if (maxCharByte < char2ByteMap[cfgb.chars.Substring(i,
1)]) maxCharByte = char2ByteMap[cfgb.chars.Substring(i, 1)];
            }


            /// sDateTime,eDateTime の設定値補正

            if (cfgb.sDateTime > cfgb.eDateTime) cfgb.eDateTime =
cfgb.sDateTime;


            /// threadCount の設定値補正
```

```
            int minwkr, minAIO;
            int maxwkr, maxAIO;
            int maxThreadsParDir;

            ThreadPool.GetMinThreads(out minwkr, out minAIO);
            ThreadPool.GetMaxThreads(out maxwkr, out maxAIO);

            /// ディレクトリ処理の並列数計算

            if (cfgb.dirThreadCount < 1) cfgb.dirThreadCount = 3;
            if (cfgb.dirThreadCount > cfgb.basePaths.Length)
cfgb.dirThreadCount = cfgb.basePaths.Length;

            /// ファイル生成スレッド数計算

            if ((cfgb.threadCount > 100) || (cfgb.threadCount <
minwkr)) cfgb.threadCount = minwkr;
            if (cfgb.threadCount > cfgb.fileCount) cfgb.threadCount =
cfgb.fileCount;

            maxThreadsParDir = maxwkr / cfgb.dirThreadCount;

            if (cfgb.threadCount > maxThreadsParDir) cfgb.threadCount =
maxThreadsParDir;

            /// 並列処理は無理
            if (maxThreadsParDir == 0)
            {
                cfgb.dirThreadCount = 1;
                cfgb.threadCount = 1;
            }
        }
        public String WritePath
        {
            get
            {
                return cfgb.writePath;
            }
        }
        public String[] BasePaths
        {
            get
            {
                return cfgb.basePaths;
            }
        }
        public String FilePrifix
        {
            get
            {
```

```
                return cfgb.filePrifix;
            }
        }
        public String FileSuffix
        {
            get
            {
                return cfgb.fileSuffix;
            }
        }
        public String Terminate
        {
            get
            {
                return cfgb.terminate;
            }
        }
        public int TerminateLen
        {
            get
            {
                return terminateLen;
            }
        }
        public int FileCount
        {
            get
            {
                return cfgb.fileCount;
            }
        }
        public int LineCount
        {
            get
            {
                return cfgb.lineCount;
            }
        }
        public int LineWidth
        {
            get
            {
                return cfgb.lineWidth;
            }
        }
        public int ThreadCount
        {
            get
            {
                return cfgb.threadCount;
            }
```

```
        }
        public int DirThreadCount
        {
            get
            {
                return cfgb.dirThreadCount;
            }
        }
        public String EncodingStr
        {
            get
            {
                return cfgb.encodingStr;
            }
        }
        public Encoding EncodingObj
        {
            get
            {
                return Encoding.GetEncoding(cfgb.encodingStr);
            }
        }
        public DateTime RandomDateTime
        {
            get
            {
                return cfgb.sDateTime.AddMilliseconds(r.NextDouble() *
(cfgb.eDateTime - cfgb.sDateTime).TotalMilliseconds);
            }
        }
        public Boolean UseRandomTimestamp
        {
            get
            {
                return cfgb.useRandomTimestamp;
            }
        }
        public int MaxCharByte
        {
            get
            {
                return maxCharByte;
            }
        }
        public int MaxCharsLen
        {
            get
            {
                return cfgb.chars.Length;
            }
        }
```

```
        public void initThreadSize()
        {
            int minwkr, minAIO;
            int newMinwkr = cfgb.threadCount * cfgb.dirThreadCount;

            ThreadPool.GetMinThreads(out minwkr, out minAIO);
            ThreadPool.SetMinThreads(newMinwkr, minAIO);
        }
        public String idx2Str(int idx)
        {
            return index2StringMap[idx];
        }
        public int idx2byte(int idx)
        {
            return char2ByteMap[ idx2Str(idx) ];
        }
        public int char2Byte(String s)
        {
            return char2ByteMap[s];
        }

        public void writeconfig(String filename)
        {
            FileStream fs = new FileStream(filename, FileMode.Create);
            XmlSerializer xs = new XmlSerializer(typeof(configBase));

            xs.Serialize(fs, cfgb);
            fs.Close();
        }

        public Boolean mkFile(String filePath, Int32 seed1, Int32
seed2)
        {
            Encoding eobj = EncodingObj;
            Byte[] buff = new Byte[LineWidth * MaxCharByte +
TerminateLen];
            Boolean ans = true;
            Random q = new Random(Environment.TickCount + seed1 +
seed2);

            using (FileStream fs = new FileStream(filePath,
FileMode.Create, FileAccess.Write))
            {
                int bufsize = 0;
                int sidx = 0;
                try
                {
                    for (int lc = 0; lc < LineCount; lc++)
                    {
                        bufsize = 0;
```

```
                        for (int wi = 0; wi < LineWidth; wi++)
                        {
                            sidx = q.Next(0, MaxCharsLen);
                            eobj.GetBytes(idx2Str(sidx)).CopyTo(buff,
bufsize);
                            bufsize += idx2byte(sidx);
                        }
                        eobj.GetBytes(Terminate).CopyTo(buff, bufsize);
                        bufsize += TerminateLen;

                        fs.Write(buff, 0, bufsize);
                    }
                    fs.Flush();

                    fs.Close();
                    fs.Dispose();
                }
                catch (Exception e)
                {
                    Console.WriteLine("{0} {1} - {2}", DateTime.Now,
filePath, e.Message);
                    ans = false;
                }
            }

            return ans;
        }
        public Boolean mkFolder(String folderPath)
        {
            Boolean ans = true;

            if (Directory.Exists(folderPath) == false)
            {
                try
                {
                    Directory.CreateDirectory(folderPath);
                }
                catch (Exception e)
                {
                    Console.WriteLine("{0} {1} - {2}", DateTime.Now,
folderPath, e.Message);
                    ans = false;
                }
            }

            return ans;
        }
        public Boolean modFile(String filePath)
        {
            Boolean ans = true;
            DateTime rt = RandomDateTime;
```

```
            if ((UseRandomTimestamp == true) && (File.Exists(filePath)
== true))
            {
                try
                {
                    File.SetCreationTime(filePath, rt);
                    File.SetLastWriteTime(filePath, rt);
                    File.SetLastAccessTime(filePath, rt);
                }
                catch (Exception e)
                {
                    Console.WriteLine("{0} {1} - {2}", DateTime.Now,
filePath, e.Message);
                    ans = false;
                }
            }
            return ans;
        }
        public Boolean modFolder(String folderPath)
        {
            Boolean ans = true;
            DateTime rt = RandomDateTime;

            if ((UseRandomTimestamp == true) &&
(Directory.Exists(folderPath) == true))
            {
                try
                {
                    Directory.SetCreationTime(folderPath, rt);
                    Directory.SetLastWriteTime(folderPath, rt);
                    Directory.SetLastAccessTime(folderPath, rt);
                }
                catch (Exception e)
                {
                    Console.WriteLine("{0} {1} - {2}", DateTime.Now,
folderPath, e.Message);
                    ans = false;
                }
            }
            return ans;
        }
    }

    class Program
    {
        static void Main(string[] args)
        {

            config cfg;
            Action[] t;
```

```
            DateTime s;
            DateTime e;

            switch (args.Length)
            {
                case 2:
                    cfg = new config("config.xml",
Int32.Parse(args[0]), Int32.Parse(args[1]));
                    break;
                default:
                    cfg = new config("config.xml");
                    break;
            }
            cfg.initThreadSize();

            //cfg.writeconfig("xconfig.xml");

            s = DateTime.Now;
            Console.WriteLine("{0} start", s);
            Console.WriteLine("  {0} Dirctories",
cfg.BasePaths.Length);
            Console.WriteLine("  {0} dirThreadCount",
cfg.DirThreadCount);
            Console.WriteLine("  {0} threadCount", cfg.ThreadCount);

            for (int bpc = 0; bpc < cfg.BasePaths.Length; bpc +=
cfg.DirThreadCount)
            {
                int dtc = cfg.DirThreadCount;
                if ((bpc + dtc) > cfg.BasePaths.Length) dtc =
cfg.BasePaths.Length % cfg.DirThreadCount;

                for (int cfc = 0; cfc < cfg.FileCount; cfc +=
cfg.ThreadCount)
                {

                    int tc = cfg.ThreadCount;
                    if ((cfc + tc) > cfg.FileCount) tc = cfg.FileCount
% cfg.ThreadCount;

                    t = new Action[dtc * tc];

                    for (int idtc = 0; idtc < dtc; idtc++)
                    {
                        for (int itc = 0; itc < tc; itc++)
                        {
                            int localbpc = bpc;
                            int localidtc = idtc;
                            int localtc = tc;
                            int localdtc = dtc;
                            int localcfc = cfc;
```

```
                            int localitc = itc;

                            String writePath = cfg.WritePath + "\\" +
cfg.BasePaths[localbpc + localidtc];

                            t[(localidtc * localtc) + localitc] = () =>
                            {
                                String fileName =
String.Format(@"{0}\{1}{2:D06}{3}", writePath, cfg.FilePrifix, localcfc
+ localitc, cfg.FileSuffix);

                                cfg.mkFolder(writePath);
                                cfg.mkFile(fileName, localbpc +
localidtc, localcfc + localitc);
                                if (cfg.UseRandomTimestamp == true)
cfg.modFile(fileName);
                            };
                        }
                    }
                    Parallel.Invoke(t);

                }

                if (cfg.UseRandomTimestamp == true)
                {
                    for (int idtc = 0; idtc < dtc; idtc++)
                    {
                        String writePath = cfg.WritePath + "\\" +
cfg.BasePaths[bpc + idtc];
                        cfg.modFolder(writePath);
                    }
                }

                Console.WriteLine("{0} {1} Dirs processed.",
DateTime.Now, dtc);
            }
            e = DateTime.Now;
            Console.WriteLine("{0} end, {1}", e, e - s);
        }
    }
}
```

おまけ

開発環境下で、SATA - USB3.0 変換器でHDDを接続し、このHDDへの書き込み処理で使うスレッド数とその処理時間変化を計測してみました。

- 変換器 - MAL-4535SBKU3

- HDD - Western Digital WD3200AAKS (320GB)
- OS - Win7 SP1 64bit版
- メモリ - 8GB
- CPU - AMD Phenom Ⅱ X6 1055T

この環境では.NET Frameworkのマネージドスレッドの最小数が6になるので、threadCountを6より小さい数値に指定してもプログラム側で6に訂正されます。 この最小数はCPUコア数と同じになるとのことです。

プログラムへの設定は以下の通り。

- サンプルのconfig.xmlを利用し、basePaths だけ4フォルダの定義に変更。※4ディレクトリにそれぞれ約3MBのファイルを1000個作成します。
- dirThreadCount には 1,2,4を指定
- threadCount、dirThreadCountは引数で変更

dirThreadCount に3を指定しないのは、指定のスレッド数を全部使い切れないためです。 例えば4ディレクトリ作成で、threadCount=25、dirThreadCount=4、だった場合は100スレッドで4ディレクトリ4000ファイルを作成します。 ですが、threadCount=25、dirThreadCount=3、だった場合は75スレッドで3ディレクトリ3000ファイルを作成したあとに25スレッドで残り1ディレクトリ1000ファイルを作成します。75スレッドで処理する訳ではないので除外しました。
※作者の力不足のせいです

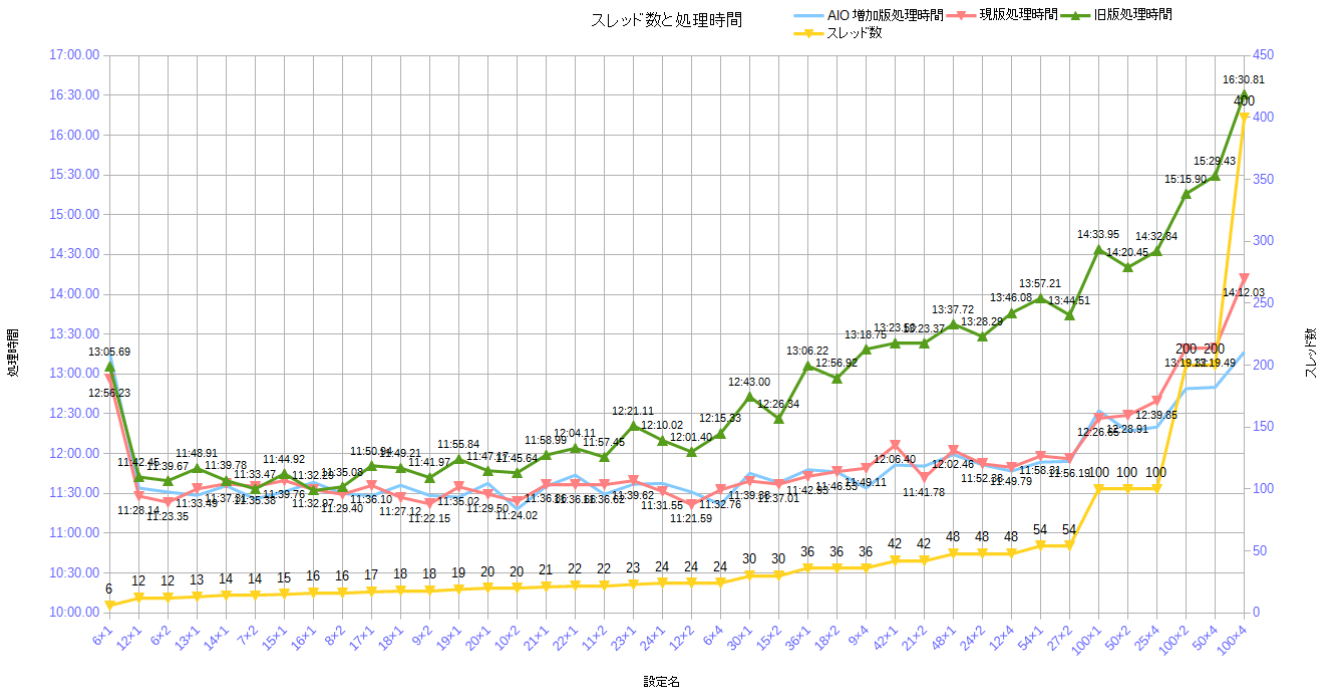
以下のグラフは、

- 旧版 - Taskオブジェクトを生成していたときのもの
- 現版 - Parallel.Invoke()を利用する現行のもの
- AIO増加版 - 現版に非同期 I/O スレッドの最大数を変更したもの(今回は初期値が6だったので、6×3=18に設定)

の各版で実行した結果をプロットしたものです。

計測結果より、現版が明らかに改善されています。おそらく、Taskオブジェクト生成のオーバーヘッドがない為と推測されます。
どの版でも12から20スレッドが処理時間を抑えられるスレッド数かと考えられます。非同期 I/O スレッドを変更してもあまり効果は見られませんでした。※有効なときの数値の組み合わせがハッキリしません。

また、現版のデバッグ時、旧版では見られなかった乱数使用時の不具合が顕在化しました。
乱数生成に使ったRandomオブジェクトはシード値に Environment.TickCount を使用します。これはマシン起動後からミリ秒単位でカウントアップされていくカウンターです。
各スレッドのファイル生成処理呼び出しタイミングが、このカウンターのカウントアップタイミングより速くなった為に、乱数値が各スレッドで同一の値となり同じ内容のファイルが大量生成されることになってしまいました。



設定名	threadCount	dirThreadCount	スレッド数	処理時間		
				旧版	現版	AIO増加版
6×1	6	1	6	00:13:05.69	00:12:56.23	00:13:15.40
12×1	12	1	12	00:11:42.45	00:11:28.14	00:11:34.10
6×2	6	2	12	00:11:39.67	00:11:23.35	00:11:31.08
13×1	13	1	13	00:11:48.91	00:11:33.49	00:11:28.65
14×1	14	1	14	00:11:39.78	00:11:37.31	00:11:35.82
7×2	7	2	14	00:11:33.47	00:11:35.38	00:11:25.93
15×1	15	1	15	00:11:44.92	00:11:39.76	00:11:31.35
16×1	16	1	16	00:11:32.29	00:11:32.97	00:11:38.39
8×2	8	2	16	00:11:35.08	00:11:29.40	00:11:30.01
17×1	17	1	17	00:11:50.94	00:11:36.10	00:11:28.15
18×1	18	1	18	00:11:49.21	00:11:27.12	00:11:36.21
9×2	9	2	18	00:11:41.97	00:11:22.15	00:11:28.47
19×1	19	1	19	00:11:55.84	00:11:35.02	00:11:27.28
20×1	20	1	20	00:11:47.17	00:11:29.50	00:11:37.67
10×2	10	2	20	00:11:45.64	00:11:24.02	00:11:18.26
21×1	21	1	21	00:11:58.99	00:11:36.86	00:11:35.47
22×1	22	1	22	00:12:04.11	00:11:36.66	00:11:43.85
11×2	11	2	22	00:11:57.45	00:11:36.62	00:11:29.39
23×1	23	1	23	00:12:21.11	00:11:39.62	00:11:36.98
24×1	24	1	24	00:12:10.02	00:11:31.55	00:11:37.64
12×2	12	2	24	00:12:01.40	00:11:21.59	00:11:31.07
6×4	6	4	24	00:12:15.33	00:11:32.76	00:11:21.79
30×1	30	1	30	00:12:43.00	00:11:39.38	00:11:45.27
15×2	15	2	30	00:12:26.34	00:11:37.01	00:11:38.19
36×1	36	1	36	00:13:06.22	00:11:42.93	00:11:47.93
18×2	18	2	36	00:12:56.92	00:11:46.53	00:11:46.27

設定名	threadCount	dirThreadCount	スレッド数	処理時間		
				旧版	現版	AIO増加版
9×4	9	4	36	00:13:18.75	00:11:49.11	00:11:34.13
42×1	42	1	42	00:13:23.50	00:12:06.40	00:11:51.48
21×2	21	2	42	00:13:23.37	00:11:41.78	00:11:50.65
48×1	48	1	48	00:13:37.72	00:12:02.46	00:11:59.35
24×2	24	2	48	00:13:28.29	00:11:52.38	00:11:51.62
12×4	12	4	48	00:13:46.08	00:11:49.79	00:11:46.90
54×1	54	1	54	00:13:57.21	00:11:58.31	00:11:53.63
27×2	27	2	54	00:13:44.51	00:11:56.19	00:11:54.26
100×1	100	1	100	00:14:33.95	00:12:26.65	00:12:32.37
50×2	50	2	100	00:14:20.45	00:12:28.91	00:12:17.11
25×4	25	4	100	00:14:32.84	00:12:39.85	00:12:20.06
100×2	100	2	200	00:15:15.90	00:13:19.32	00:12:49.00
50×4	50	4	200	00:15:29.43	00:13:19.49	00:12:50.07
100×4	100	4	400	00:16:30.81	00:14:12.03	00:13:16.64

Windows, text, tool